一．Java 中的垃圾是什么？

答：从垃圾回收的根出发是否可见。根：jvm认为一定有用的东西。

二. 什么是垃圾回收的根？

垃圾回收的根节点。

1. 局部变量

2. 静态变量

3. Native 方法所引用的对象

4. 活动线程，等待中的Monitor(wait，notify, synchronized);

三. 垃圾回收算法：

1 .Mark and Sweep 算法(标记清除算法)。

标记清除算法缺点： 碎片化严重。

2. Compact算法(压缩算法); 缺点：

3. Copy 算法（复制算法）； 缺点：

四. Java 分代垃圾回收算法？

基础假设：大部分对象只存在很短的时间。

将内存分为新时代，老生代，持久代。

将新生代分为 Eden, Survivor1,Survivor2 区

新生代存活一定次数会被转入老生代。默认15次

Major/Full GC 对老生代做GC

新生代采用copy算法。

老生代采用Compact 算法。

五：新生代的Minor GC

Eden

Survivor1

Survivor2

参数配置

-XX:NewRatio 老生代（666）/新生代（333）比例，默认2

-XX:SurvivorRatio Eden/Survivor比例，默认是8 （8:1:1）

-XX:MaxTenuringThreshold 新生代转至老生代阈值，默认15

Permanent Generation：持久代。

放置ClassLoader 读进来的Class，除系统 Class外。

放置String.intern 后的结果。

解决OutOfMemoryError: PerGen Space方法。

使用-XX:MaxPerSize =1024M调整。

PermGen Space vs Metaspace

Java 1.8 使用使用Metaspace 取消PermGen Space

String.intern 的结果被放入堆。

Metaspace 默认不设置限制，使用系统内存

六： 谈谈垃圾回收机制？

七： 垃圾回收在什么时候运行？

1.内存满了，分配内存失败时会运行垃圾回收。

2.当调用System.GC 方法时。

八：垃圾回收对什么对象进行回收？

九：垃圾回收算法对内存划分成了哪些区域？

新生代：Eden Survivor1 Survivor2 （copy）

老生代：（compact）

持久代：（）

十：Java 垃圾回收的调试？

获取信息：

-verbose：gc

-XX:+HeapDumpOnOutOfMemoryError



Spring Actuator

查看信息：

官方：visual ， jmap。

第三方：Eclipse Memory Analyzer

在线： gceasy.io fastthread.io

JAVA 内存模型

1. Java 出现多线程安全问题的原因？

多线程执行乱序，

内存的可见性。

1. Java Memory Model

定义： happens-after关系

如果操作执行顺序具有先后性。

那么后执行的操作能够看到先执行的操作的结果。

规定以下操作必须保证happens-after 关系

Unlock 发生在lock之前。

写volatile发生在读volatite 之前

线程start() 发生在线程所有动作之前。

线程中所有操作发生在线程join之前。

构造函数发生在finalizer 开始之前。

传递性：happens-after关系满足传递性 A>B B>C =>A

————————————————————————————————————

作为一种阅读的方式了解下jvm的工作原理

五．生产者消费者模式实现

**package** com.provider.consumer;

**public** **class** Demo{

**public** **static** **void** main(String[] args) {

Resource r=**new** Resource();

Producer p=**new** Producer(r);

Consumer c=**new** Consumer(r);

Thread t1=**new** Thread(p);

Thread t2=**new** Thread(c);

t1.start();

t2.start();

}

}

**class** Resource {

**private** String name;

**private** **int** count =1;

**private** **boolean** flag=**false**;

**public** **synchronized** **void** set(String name) {

**if**(flag)

**try** {wait();}**catch** (Exception e) {}

**this**.name=name+"--"+count++;

System.***out***.println(Thread.*currentThread*()+"...生产者..."+**this**.name);

flag=**true**;

**this**.notify();

}

**public** **synchronized** **void** out() {

**if**(!flag)

**try** {wait();}**catch** (Exception e) {}

System.***out***.println(Thread.*currentThread*()+"...消费者..."+**this**.name);

flag=**false**;

**this**.notify();

}

}

**class** Producer **implements** Runnable{

**private** Resource res;

**public** Producer(Resource res) {

**super**();

**this**.res = res;

}

@Override

**public** **void** run() {

**while**(**true**) {

res.set("+商品+");

}

}

}

**class** Consumer **implements** Runnable{

**private** Resource res;

**public** Consumer(Resource res) {

**super**();

**this**.res = res;

}

@Override

**public** **void** run() {

**while**(**true**) {

res.out();

}

}

}

六．JVM 执行new语句创建一个对象时，内部的运行过程是怎样的 ？

1. 所有的类都是在第一次被使用时，动态加载到JVM中。当首次创建类型为Dog的对象时，或者Dog类的静态方法首次被调用时，或者静态属性域首次被访问时，java解释器查找classPath，定位到Dog.class文件。   
  
2. 载入Dog.class文件，生成一个Class类型对象，所有有关的静态初始化动作都会执行：如静态代码块，静态成员属性。 并且这种初始化动作只在Class对象首次加载时候进行一次。   
  
3. 当用new Dog()创建对象时，首先JVM在堆heap上为Dog对象分配足够的存储空间。   
  
4. 存储空间清空，自动将Dog对象中的所有基本类型数据都设置成了默认值，对象引用被设置为null。   
  
5. 执行所有在字段定义处的一些初始化操作。   
  
6. 调用构造器方法。（没有继承）